VEVI: A Virtual Reality Tool For Robotic Planetary Explorations

Laurent Piguet [1]
Terry Fong [1], Butler Hine, Phil Hontalas, Erik Nygren [2]

NASA Ames Research Center
Intelligent Mechanisms Group
MS 269-3
Moffett Field, CA 94301

1. Abstract

The Virtual Environment Vehicle Interface (VEVI), developed by the NASA Ames Research Center's Intelligent Mechanisms Group, is a modular operator interface for direct teleoperation and supervisory control of robotic vehicles.

Virtual environments enable the efficient display and visualization of complex data. This characteristic allows operators to perceive and control complex systems in a natural fashion, utilizing the highly-evolved human sensory system.

VEVI utilizes real-time, interactive, 3D graphics and position / orientation sensors to produce a range of interface modalities from the flat panel (windowed or stereoscopic) screen displays to head mounted/head-tracking stereo displays. The interface provides generic video control capability and has been used to control wheeled, legged, air bearing, and underwater vehicles in a variety of different environments [1].

VEVI was designed and implemented to be modular, distributed and easily operated through long-distance communication links, using a communication paradigm called SYNERGY.

2. Introduction

2.1 Background

Mission of the IMG

The objective of the Intelligent Mechanisms (IM) group is the systems investigation of intelligent mechanisms. The research is focused by the task of building intelligent mechanisms, rather than being driven by a specific technological bias. Pursuant to this focus we have concentrated on architectures for intelligent mechanisms, including software architectures, advanced processors, sensor processing (including vision, tactile, and proximity sensors) and user interfaces [9].

Focus

The application focus of this group is driven by the relative importance of this technology to NASA missions, in three specific areas:

(1) construction and exploration tasks on planetary surfaces.

(2) low overhead operations for orbital missions.

(3) using undersea vehicles as analogs of space vehicles.

The products of the IM group are advancements in the ability to accomplish a NASA mission.

Current Research

Crucial to the success of intelligent mechanisms is suitable computational systems and methods for robustly evaluating and handling faults. Additionally, operational needs in unstructured or changing environments requires appropriately constructed systems architectures incorporating multiple sensor data, intelligent software and user interfacing. At this time, therefore, our work is directed towards:

- Advanced computing incorporating high performance processing and software architectures.
- Sensor processing using visual, tactile and proximity sensors.
- User interfaces incorporating virtual environments and telepresence.
- Systems integration of components into demonstrable applications.

Facilities

The IM laboratory is located in the Automation Sciences Research Facility (ASRF) at the NASA Ames Research Center (ARC). This facility houses the Computational Sciences

1 Recom Technologies, Inc., San-Jose, CA.

2 Massachusetts Institute of Technology, Cambridge, MA.

Division (Code IC).

The majority of the Intelligent Mechanisms Group's research is performed in the IM Laboratory at Ames Research Center. Additionally, the group conducts collaborative work with other ARC research groups and external organizations. The IM group currently has access to manipulator arms, mobile platforms, undersea vehicles, and high performance workstations running various user interface environments.

Computational Architecture

In [1], we have described the computational architecture used to fulfill the requirements of our typical systems (ARCA: Ames Robotic Computational Architecture).

In summary, the architecture provides solutions for the problems linked to different issues:

- Multiple data streams (sensory, command, knowledge) from multiple sources.
- Need to deal with synchronous processing (common clock, regular execution, strict execution schedule), loosely-synchronous processing (similar to previous, but without tight synchronization), and asynchronous processing (event-driven or free-running processes without synchronization).
- Multiple time delays (continuous to minutes).
- Application specific processor thoughput requirements (0.5 MIPS - 500 GIPS).
- Synchronization of communications between multiple modules (standardized communications).

The present paper describes an approach to the implementation of systems onto ARCA.

3. Design

3.1 Introduction

This paper describes the version 3.0 of VEVI. Previous versions have been used for a variety of tasks, though were implemented on a case-to-case basis, with application-specific code. The primary motivation for the development of VEVI 3.0 was to simplify the maintenance of application-specific code, and also to provide a better platform for future developments.

Past applications of VEVI include remote control of the following different systems:

- TROV (Telepresence Remotely Operated Vehicle) - an underwater vehicle remotely controlled from California to the Antarctic.
- MEL (Mobile Exploration Landrover) - our mobile rover, operated and controlled locally.
- Dante was a walking mechanism developed by Carnegie Mellon University, and remotely controlled from a site local to the volcano it was exploring, with sites over the country using VEVI to display telemetry information and terrain maps.
- Marsokhod is a planetary rover prototype which has been operated remotely in the Mohave desert, the Kamchatka peninsula and in Moscow from California.

Based on those wide cases, we came up with a clear understanding of the requirements of most of the applications we are developing.

3.2 Requirements

3.2.1. Distribution

In order to maintain a standardized base, from which extensions may be developed, we desired that version 3.0 be easily customizable by end-users. At the same time, we wished to maintain interoperability by restricting access to the underlying structure. We therefore sought a design which would allow flexible customizing without a requirement for the kernel source code.

3.2.2. Flexibility

Based on the provided tools, a user must be able to interface with VEVI using an easy but extensible interface. Instead of provided access to the base tools, which would increase system complexity, we decided to follow an alternate approach, in which the user can extend the capability of the basic system, to fulfill any specific need. With this approach, the system remains simple but provides possibilities for further implementations.

In a different case, the user might want to bypass the library of provided tools and use his own. This is made possible as an alternative.

Finally, in previous versions, modifications were made to the software in order to adapt the system to different situations and vehicles. The new approach provides with a way to define the environment through configuration files, which allows much better flexibility and turn-around time between projects.

3.2.3. Modularity

In order to achieve the specified goal, it was necessary to utilize a very modular approach. Modularity is attained by utilizing an object oriented paradigm. Basically, every entity in the environment is viewed as an "object". Object then can interact and communicate through messages. This event-driven approach provides us with a stateless system which allows add-ons and removal of parts of the system without destruction of the whole. Also, it allows separate development of modules, which then can be added to the core and provide additional capability, without touching the existing base.

With this approach, a user can write an entire set of new objects, which will extend the system with needed specific functionality, without complicating the provided base capabilities.

3.2.4. Standard communications

Since our system is articulated around ARCA [1], and

includes a whole set of disparate platforms and computing architectures, which communicate through a backbone, it is clear that we needed to standardize somewhat the communication protocols.

Constraints linked to remotely operated vehicles (sometimes on a different continent, or planet), as well as the capability to be as close as possible of an existing standard made us choose to define communication tools based on the Internet Protocol (IP) as our standard. So far, we have used three existing networking tools based on an IP layer: TCA [2;3], TCX [4], TelRIP [5] and NDDS [6].

Although most of our past missions were conducted using one or more of those standards, it clearly showed that we need to have the capability to adapt to a number of those tools.

External collaboration for past and future missions, as well as constraints linked to data rates and bandwidth are a determining factor for the choice of one or the other standard.

3.2.5. Execution speed

Based on previous experience, it is necessary to maintain a frame rate at the renderer's level that is greater than 10 frames per second (fps). At lower frame rates, the user experiences significant loss of the immersion provided by Virtual Environments.

To comply with such a tight constraint, especially on lower-end platforms, the rendering node will contain as little "intelligence", or application-specific data and algorithms as possible. Nodes hanging off the backbone (Internet) will be used in the cases where such knowledge is required and will communicate directly with VEVI.

Also, a VEVI might not reach the necessary frame-rate needed to efficiently process data sent at much higher rates. For this reason, we decided that an external process, known as CommTask, would be responsible for dealing with different data flows, buffering and queuing of data. Also, CommTask will be able to adjust the way it delivers messages, based on directives from VEVI directly. For example, VEVI might request CommTask to only send the most "urgent" message, not to send messages regarding one object, or to size down the number of messages made available at each polling session.

3.2.6. Distributed environment

One of the group's goals is to develop technology which allows scientists and the public to access missions sites and data. Therefore, we want to have the possibility for people to interface with vehicles, tools and data gathered using their own low-cost platform at the office, or at home, throughout the country.

We also want to be able to provide multiple people separated with the possibility to interface and collaborate through multiple Virtual Environment with the remote vehicle.

3.2.7. Portability

In order to reach the previous goal, we need to have the possibility to run the software and the main rendering tool on PC platforms, as well as high-end graphic machines.

Links with the backbone will be created through alternate communication protocols (serial over a modem line, ISDN) through a translator which will convert data to the communication protocol used on SYNERGY[1].

3.2.8. Data types

Complex and mixed systems present two different kinds of data that we need to be able to deal with:

- Streams: the data flows at any rate. Whomever is interested in it will obtain the latest update and process it. Some updates may be lost, multiple updates may be received between lookups, the order is not important. Such message do not queue up in case the lookup rate is lower to the sending rate.
- Sequences: the data is sent out in a particular order, which needs to be respected. Every message needs guaranteed delivery, in the right order. In this case, buffering and queuing become important issues, especially when the provider outputs information at a much higher rate than the consumer can process it. Sequences are typically higher-level commands, and therefore aren't sent out at very high rates over an average period of time, but the need to process and store the information remains a priority to handle a higher rates than usually provided by VEVI.

The CommTask previously mentioned will be used to process the data and store it, independently of the frame rate of the rendering node. It will then deliver the information, based on its properties, as well as the requirements of VEVI, on every request. With this approach, all VEVI will have to do is simply collect whatever information is delivered, knowing that it correspond to previously defined requirements. A great deal of computation is therefore avoided in the rendering node, thus allowing better frame rate in the Virtual Environment.

3.3 Summary

- Object-oriented approach
- Messages between objects
- Based on a IP backbone for communications
- Rendering at > 10 fps
- Configuration File
- CommTask for buffering and conversion between vehicle-specific info and VEVI generic info, as well as dealing with information delivery.
- Modules communicating and operating of the backbone

1. Logical representation of the system as a whole, including the different nodes hanging off the backbone (see Fig. 1)

4. Implementation

4.1 Introduction

Since SYNERGY is by definition an evolving architecture, it is difficult to represent a general configuration. However, we will in the next paragraphs refer to one fairly simple and coherent application, with its proper configuration. SYNERGY, however, can be extended and modified, and is not limited to the example presented herein.

Several main elements can be found in any implementation of SYNERGY:

- VEVIs (rendering nodes), or nodes which provide 3D interactive graphics.
- CommTasks which take care of communications between VEVIs and the rest of SYNERGY
- Special purpose nodes, which represent any particular computing process (simulators, planners, etc.), generally located on machines dedicated to those processes.
- Vehicle nodes, which are the symbolic representation of a vehicle, seen from the rest of SYNERGY (through the on-board controller).

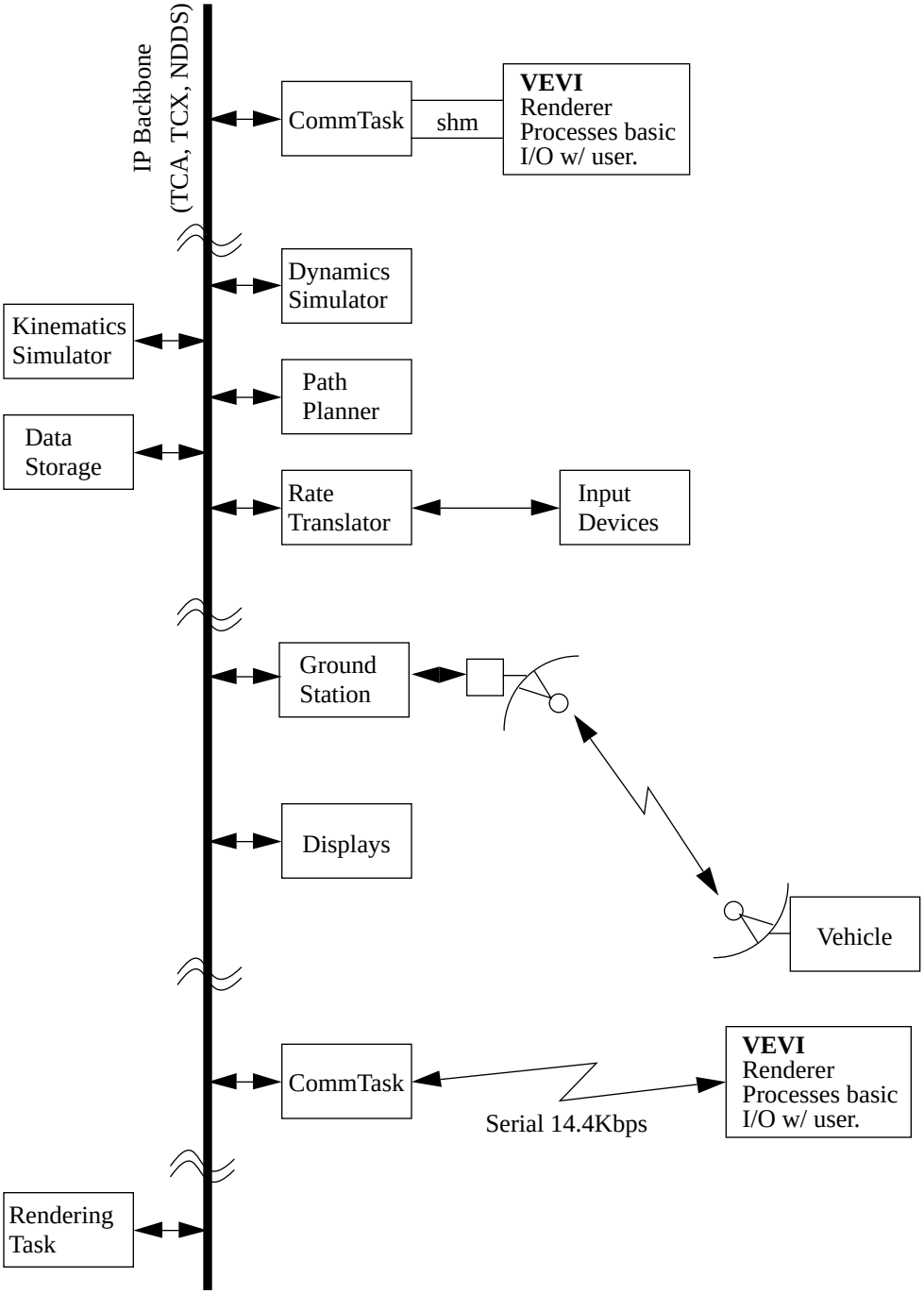

Fig. 1: SYNERGY: overall communication layout.

- Communications, to link all the nodes.
- Platforms. All nodes can run on different platforms.

4.2 VEVIs

4.2.1. Description

When referring to the overall system, we call VEVI a “rendering node”. Its main purpose is to give an interactive 3D representation of a Virtual Environment to a user, through multiple input and output devices.

Multiple VEVIs can exist at the same time in SYNERGY, representing for each user chosen information. It is then possible to have several people share an environment and obtain informations at the same time. Each user can focus on his primary zone of attention, by acting on the sensors and commands at his/her disposition.

It is important to understand that VEVI can be used as a simulation system, bringing recorded data into a realistic 3D interactive universe with which the user can interact, or as a “3D window” displaying the current state of multiple complex systems as they operate in a remote environment. The difference between those two situations is simply based on the existence or not of real-time data sent from the systems on SYNERGY.

4.2.2. Structure

VEVI follows an object-oriented approach. Within its main kernel, each entity of the world, represented or not in the Virtual Environment, exists as an object loaded from a configuration file.

Objects communicate between each other with messages, which can contain data. All interactions between objects are defined by message passing, creating a stateless system (see Fig. 2).

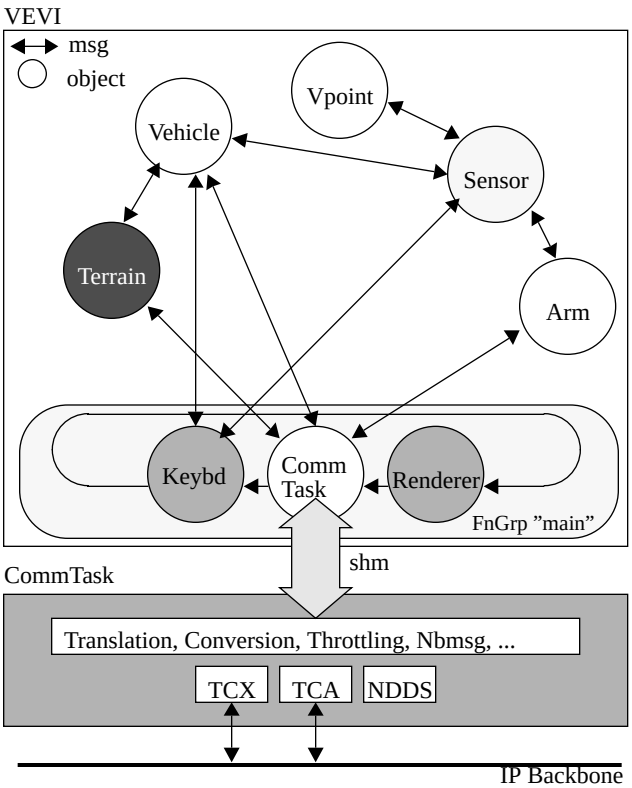

Fig. 2: VEVI & CommTask: objects, message flows

4.2.3. VCF file format

Description
The VEVI Configuration File format was designed to allow the full definition of objects present in the environment in a file read at initialization. Advantages are the added flexibility of being able to change this file, rather than modify the source code and re-compile.

In order for this concept to be useful, the configuration language must provide a structure that allows implementation of any kind of data which might be useful for the description of an object.

Configuration files are preprocessed using the standard C preprocessor, and therefore allowing macro definition and multiple file inclusion.

Structure
The file format contains tag names followed by arguments. Each argument can either be a string containing data, or a sublevel, surrounded by brackets “{...}“.

The tag names at the upper-level always refer to objects. Deeper levels are defined differently, depending on each object.

Example
This example describes the vcf representation of a vehicle with an on-board manipulator arm.

```
wtk_object { 1
    name “theVehicle” 2
    filename “theVehicleModel.nff” 3
    scale 1.0 4
    initpos { z -50.0} 5
    attach_sensor “A_Sensor” 6
}

manipulator {
    name “theArm”
    raff_file “theArmD-Hdescription.raff” 7
    scale 1.0
    initpos {
        frame “theVehicle” 8
        x 75.0 y 6.0 z 0.0
        ex -90.0 ez 180.0 9
    }
    father “theVehicle” 10
    attach_sensor “Another_Sensor”
}
```

1. Type of object.
2. Name of this instance.

3. File that contains the geometric data.

4. Scaling factor.

5. Initial position. If not mentioned, the reference frame is the World frame.

6. We attach a sensor, defined by the name of the created object.

7. For manipulator arms, file containing all the parameters necessary for its creation.

8. In initpos, allows the specify any pose with respect to the frame of any other defined object of the environment.

9. Orientations can be specified as quaternions, or as fixed (XYZ) angles (equivalent to Euler ZYX).

10. Establishes a hierarchy. The named object becomes the father of the current one.

4.2.4. Objects

Definition
Each instance of an object is defined as a data structure. The data of each object is private and can be accessed by the object alone. It contains a private data area, which is defined when creating the object library, and pointers to functions used at creation, to handle messages, and to destroy the object (see Fig. 3).

Creation
At creation time, the private data area is created, the vcf entry containing the description of the object is processed and initializations are performed.

Handlers
Handlers define the behavior of an object when receiving messages. Upon receival of a message, the called object will invoke its "handle" function, which will interpret the message, its data, and generate predefined action. Finally, a status message is send back to the originator.

Destruction
The destructor handles mainly memory issues, and clean-up of any data used by the object.

Messages
Messages can contain complex data, that is processed upon receipt. Each message is composed of an ID, and optionally some data.

Message in this implementation are blocking, which means that the remaining processes will be executed only upon receipt of execution of the message sent.

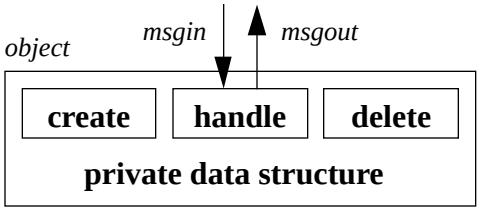

Fig. 3: Structure of an object

4.2.5. Examples

Here are some of the currently implemented objects:

"Function Groups"
In order to achieve a degree of loose synchronism, a particular object, called a "function group", had been implemented. They have the capability of containing other objects, to which they send predefined messages each time they are themselves invoked. Function groups can loop.

"Graphical object"
Defines every 3D representation of objects in the universe. Based on CAD description, those geometric objects have multiple parameters that can be modified through messages (position, orientation, color, scale, hierarchy, etc.). Complex mechanisms can be created by assembling multiple objects and assigning hierarchy relationships.

"Terrain"
Since most of our missions involve operation on a priori unknown terrain, we felt a need for the creation of a terrain object. Based on a height-field description of the terrain, we can represent it in the environment.

Using sensory readings received from a vehicle, we are able to create a terrain object which will represent exactly the sensed terrain and provide an intuitive interface for the user to analyze its features.

"Viewpoint"
Viewpoint objects represent cameras, through which the user can look. Multiple viewpoints can be present within one environment, assigned eventually to several windows. Viewpoints can have behaviors defined that lock them looking at a particular object, or in a certain direction. Sensors can also be attached to viewpoint to allow the user to "look-around" interactively.

"Manipulator arm"
Other interesting feature, manipulator arm objects allow construction and representation of complex serial manipulator arms, based on a modified Denavit-Hartenberg definition of their parameters [11,12], as well as models composing each of their parts. Once a manipulator object is created, it is possible to attach a sensor to it, and manipulate the arm's end-effector intuitively, using a sensor.

A manipulator object can also receive an outside telemetry stream which will make it represent the current joint configuration of the arm aboard a vehicle.

"Keyboard"
A keyboard object links a key to an output message and destination. As a user presses a key, the corresponding message is send to its destination, triggering some action.

Keyboard mapping can be easily modified simply by changing the binding description in the VCF (VEVI Configuration File) file.

"Renderer"
This object, which is mandatory in most cases, takes care of the actual rendering of the scene. Currently, all our work has been developed using WoldToolKit, from Sense8 Corp, Sausalito. However, it is important to notice that this object can be interchanged for any other renderer if the need was felt. In particular, this might be an interesting approach for platforms not supported by actual versions of WTK, or in the case where this product is not available anymore.

"CommTask"
When VEVI is running in conjunction with complex systems on SYNERGY, the CommTask object is mandatory. It is responsible for the transactions between the CommTask process, running as a separate process and VEVI. This object pulls the information made available by the CommTask process and updates the objects affected in the Virtual Environment.

In cases where VEVI is used to display a simulation of the environment, without any interaction with the rest of SYNERGY, CommTask is not necessary.

4.3 CommTask

The CommTask process is an important part of SYNERGY. It was designed and implemented to respond to several needs.

a) In complex systems with multiple producers of data, and in the absence of tightly-synchronized processes, it is almost impossible to avoid differences in data rates. This has for consequence to create data accumulation and, if not processed properly, dangerous queuing of information. This is simply due to the fact that a consumer might not be able to receive the data and dispose of it in time before a new update arrives.

In our situation, the main consumer of data, VEVI (rendering node), will typically be running at rates close to 10 Hz, which would place us at risk of encountering a data accumulation situation very easily.

b) Since a user might not always be able to run VEVI on a machine with access to the IP backbone, it is necessary to provide an interface process that will communicate between them with alternate protocols. In most cases, we will use a shared memory connection to transfer data, but we can also use a serial connection.

c) We also don't want VEVI to worry about sorting through the masses of messages circulating to find out which ones are interesting to the user. A separate process, dedicated to this task can therefore take care of this task. Also, it will permit to define the number of messages passed at each request, and make the distinction and accounting of data types.

4.4 Nodes

Due to the architecture of SYNERGY, it is possible to implement multiple communicating nodes. Each one of those node has a particular function, and understands certain messages.

When new capabilities are expected, or in a case where some new tool is developed, one can add a node to the backbone, and simply define a message flow that will use the new node to process data from the environment. This approach allows the system not to be limited by present performance and tools.

Also, different nodes can be executed on different CPUs, for peak performance. The resulting latency might become a problem for certain applications, but in most cases, we are dealing with asynchronous processes and latency is not a handicap. If we need to avoid latency, we will try to group processes that linked tighter synchronism in order to reduce delays.

Since the system is stateless, it is possible to replace nodes and interchange them. Also, in some cases, a missing node will simply remove some extended capability but won't prevent the execution of the experiment. Each node has its own development curve, and the evolution of the whole is created by simultaneous evolution of each one of its parts, but isn't held back by one of them in particular.

In cases where several processes use the same algorithms, this approach allows to maintain uniqueness of data and reproducibility of data, since a unique process will deliver results to any consumer that requires it.

4.4.1. Vehicles

A vehicle can be considered as a node, since it processes data and broadcasts information about its state to any interested party within SYNERGY. The different sub-systems of a vehicle can be implemented as different nodes, each one of them hanging of the backbone, or as parts of the main process. Difference in the implementation have to be determined from case to case.

4.4.2. Special purpose nodes

Simulators

Simulators are frequent nodes that one might want to implement on SYNERGY. For example, in the case of a kinematics simulator: a user, in the environment is controlling an arm by guiding its end-effector. Depending on whether or not we want to have the real arm reflect the user's command, we can direct messages to a simulator or to the real arm.In the first case, when the simulator receives a vector of motions in the cartesian space, it translates it into joint velocities, which are returned the virtual representation of the arm for update.

In the latter case, the vector would be sent directly to the arm controller, which then would use the same simulator to generate joint velocities for the real arm.

Depending on the availability, or the appropriateness of using the real arm, we have the system behave in two different ways, yet it is transparent to the user

.

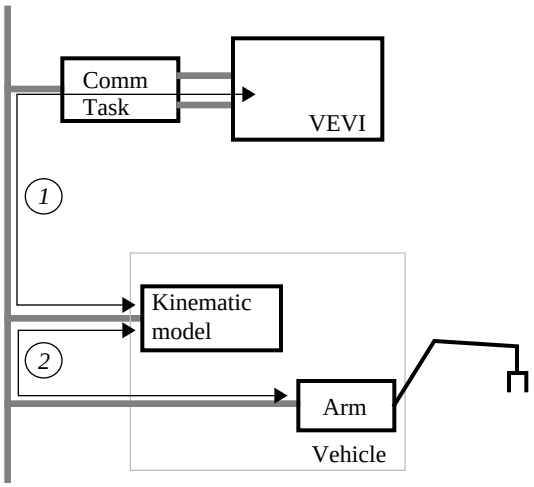

Fig. 4: Data flow in SYNERGY with a simulator node. First, the communication is established between the virtual representation of an arm and the kinematic model. Then, the real arm uses the same kinematic model to reproduce commands sent from the simulation.

Translators
In some cases, it might be necessary to translate data coming from a node before sending it to another. A translator node could achieve this, and allow to maintain the current structure of both the provider and the consumer of data.

Renderer
Some complex computations might be necessary in order to create better representation of the data. For this purpose, we have nodes which execute as a task on a separate machine to create photo-realistic views of the data gathered. Since this is a time-consuming activities, those nodes are generally routed to high-performance machines on a remote site.

5. Applications

We will shortly present some of the vehicles controlled using the described architecture. In all those cases, the virtual equivalent of a real vehicle was updated by telemetry updates received through SYNERGY.

5.1 MEL

The Mobile Exploration Landrover (MEL) has been under development since July 1992. The vehicle has two independent drive wheels and a variety of sensing devices (differential GPS, magnetic compass, I.R. sensor, ultrasonic sensors, stereo pan/tilt cameras, etc.), as well as a wireless Ethernet for communications with SYNERGY.

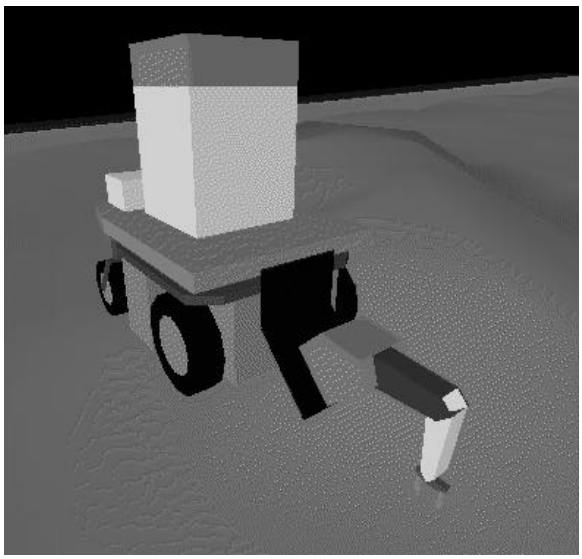

Fig. 5: MEL in the virtual world.

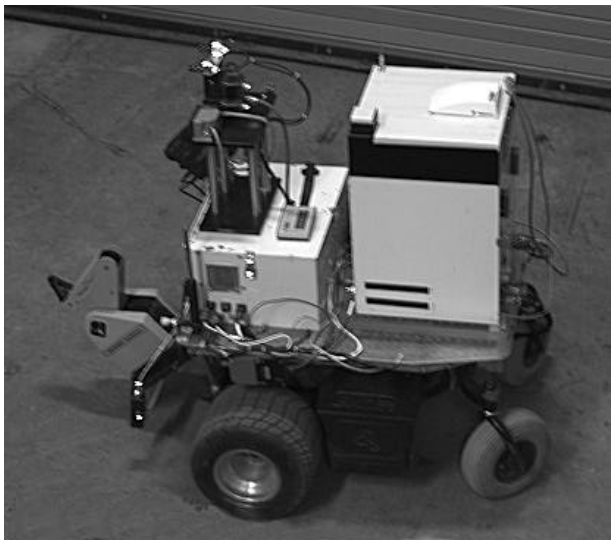

Fig. 6: The real MEL, with the arm, sensors and a pan/tilt stereo camera.

5.2 TROV

At the end of 1993, TROV (Telepresence Remotely Operated Vehicle) was deployed under the sea ice near McMurdo Science Station, Antarctica and teleoperated from a control station located at NASA Ames, California, as well as several other locations in the country. Antarctica, like Mars, has remote and hostile locations that are difficult for humans to explore. The purposes of this mission were to explore below the surface of McMurdo; conduct a benthic ecology survey; perform a study of human teleoperation performance and demonstrate virtual environments based teleoperation techniques [7,8].

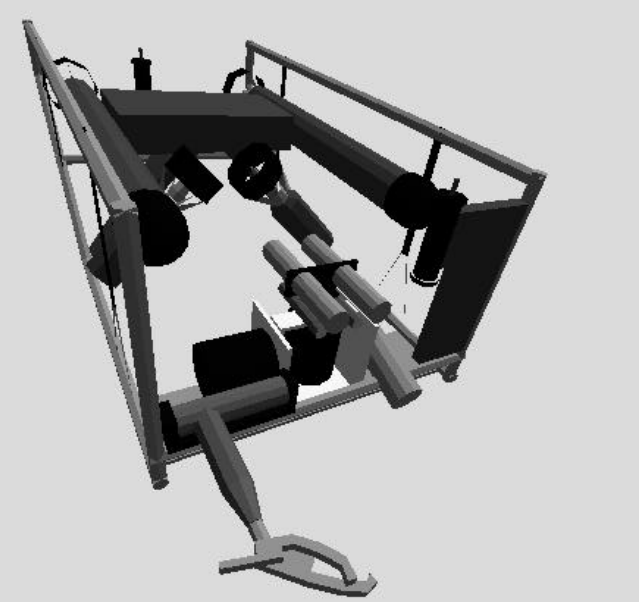

Fig. 7: TROV in the virtual world.

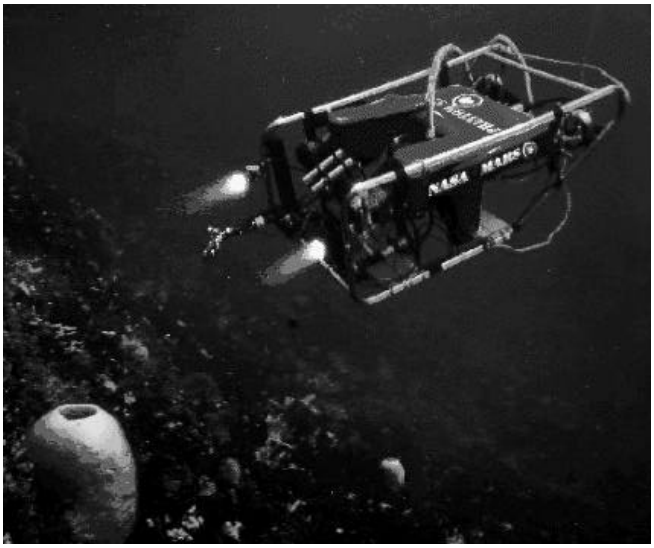

Fig. 8: TROV under the ice in the Antarctica

5.3 Dante

Dante, a frame walker robot, was sent into Mt Spurr, in Alaska in July 1994. It was controlled via satellite and Internet connections by a team with representatives from NASA, Carnegie Mellon, the Alaskan Volcano Observatory, and other government, university and private organization [10].

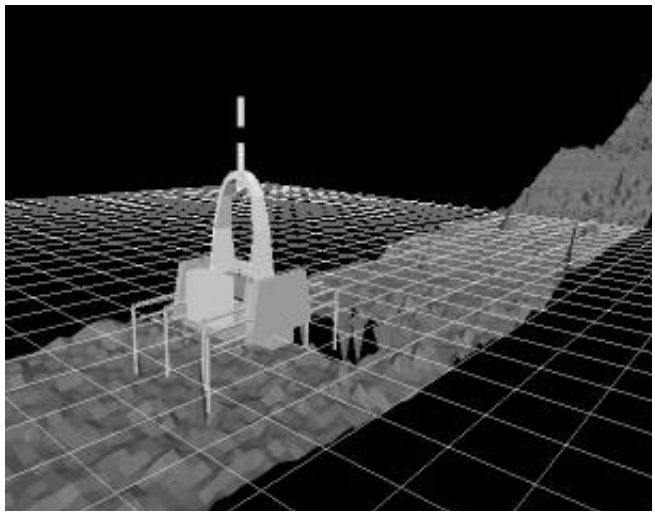

Fig. 9: Dante in the virtual environment. Along with information about the forces and torques on the limbs and the tether, terrain maps went scanned using Dante's on-board laser scanner, and shipped through the Internet to be displayed in the virtual world. Based on this information, operators of the vehicle could take better routing decisions, based on the configuration of the terrain.

Fig. 10: The real Dante on a transition.

5.4 Marsokhod

Originally designed in Russia to do inspection on the site of Tchernobyl, Marsokhod (Mars Walker in russian) is an articulated 6-wheeled rover. Its excellent capacities on rugged terrains made it an ideal candidate for future planetary missions. It is anticipated that one of those rovers will be sent to Mars sometime around 1998.

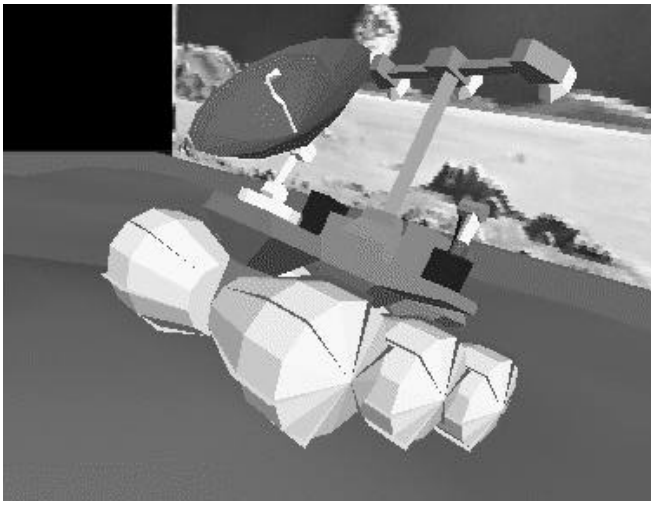

Fig. 11: Marsokhod in VEVI. Terrains created from real data, as well as texture maps contribute greatly to the realism of a scene.

Fig. 12: The real Marsokhod.

5.5 Ranger

Ranger is a free-flying satellite servicing robot. It has two 7-dof manipulator arms, a camera arm and a grapple arm. This vehicle is currently developed by the University of Maryland, Space Systems Laboratory. We will provide an alternate control interface for the visualization of scientific and telemetry data, as well as a fully-immersive operator interface using VEVI.

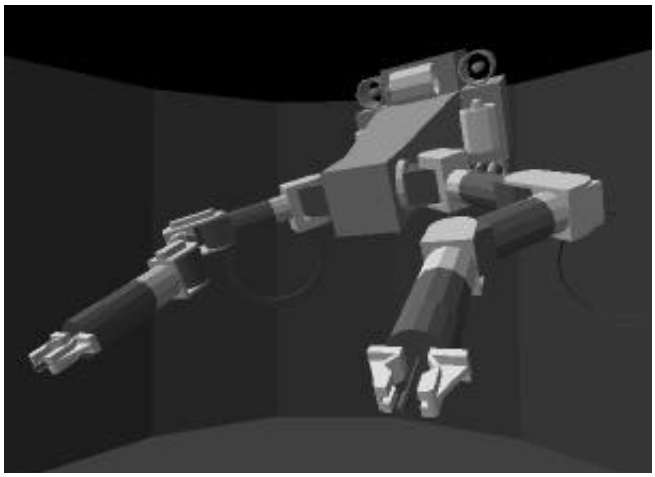

Fig. 13: The virtual Ranger in its current development stage.

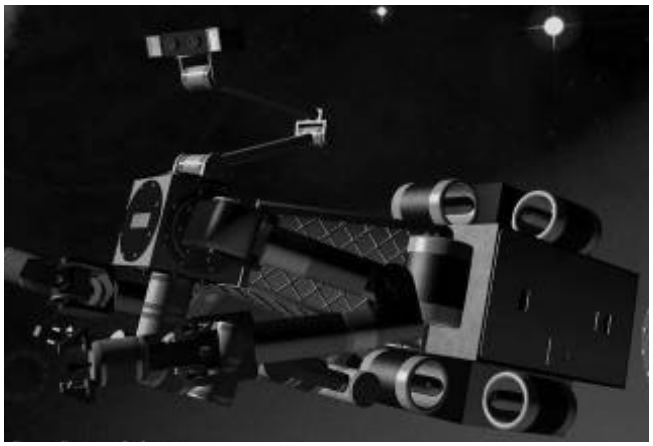

Fig. 14: A view of a rendered image of a fully equipped Ranger. Notice the camera arm, and the stowed grappling arm.

6. Conclusion

This paper presents an overview of VEVI as well as the complete networking concept SYNERGY. Version 3.0 is currently under development and more capabilities should be added to it in the next months.

So far, VEVI 3.0 has reached our expectations. The configuration file approach has permitted rapid prototyping and implementation of various environments, without showing any major limitation. Detailed benchmark haven't been performed so far, but a performance loss, which could be expected because of the extension of generality, hasn't been yet demonstrated.

Further reference and information, consult our WWW server: http://maas-neotek.arc.nasa.gov

7. Acknowledgments

This work was partially supported by NASA RTOP #233-02-04, #233-03-03, program manager D. Lavery.

We wish to thank Aaron Kline, UC Berkeley, for his programming effort.

8. References

[1] Fong, T.W., "A Computational Architecture for Semi-autonomous Robotic Vehicles", AIAA Computing in Aerospace 9 Conference, October 19-21, 1993, San-Diego, CA.

[2] Simmon, R., Goodwin, R., Fedor, C., Basista, J., "TCA Task Control Architecture, Programmer's Guide to Version 7.3", Carnegie Mellon University, School of Computer Science / Robotics Institute.

[3] Lin, L. Simmons, R., and Fedor, C., "Experience with a Task Control Architecture for Mobile Robots", CMU-RI-TR 89-29, Robotics Institute, Carnegie Mellon University, Pittsburg, PA, December 1989.

[4] Fedor, C., "TCX Task Communications", Carnegie Mellon University, School of Computer Science / Robotics Institute.

[5] Wise, J. D., Ciscon, L., "TelRIP Distributed Applications Environment Operating Manual", Rice University, Houston, Texas, 1992. Technical Report 9103.

[6] Pardo-Castellote, G., Schneider, S., "The Network Data Delivery Service: A Real-Time Data Connectivity System", Proceedings of the International Conference on Robotics and Automation, IEEE, May 1994.

[7] Hine, B., et al. "The application of Telepresence and Virtual Reality to subsea exploration", The 2nd Workshop on Mobile Robots for Subsea Environments, Proc. ROV'94, May 1994, Monterey CA.

[8] Stocker, C. et al., "Use of Telepresence and Virtual Reality in undersea exploration: 1993 Antarctic Telepresence experiment, AAAI Workshop on AI Technologies for Environmental Applications, July 1994, Seattle, WA.

[9] Fong, T.W., Hine B.P., and Sims, M.H., "Intelligent Mechanisms Group: Research Summary", NASA Ames Intelligent Mechanisms Group (IMG) internal document, Moffett Field, CA. January 1992.

[10] Wettergreen D., Thorpe, C., Whittaker, W., "Exploring Mount Erebus by Walking Robot", Robotics and Autonomous Systems 11, 1993, pp. 171-185, also in Proceedings of the Third International Conference on Intelligent Autonomous Systems, February 1993, pp. 72-81.

[11] Denavit, J., Hartenberg, R.S., "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices", Journal of Applied Mechanics, pp. 215-221, June 1955

[12] Paul, R.P., "Robot Manipulators, Mathematics, Programming and Control", The MIT Press series in artificial intelligence, 1986.